In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.simplefilter(action='ignore')
pd.options.display.max_columns = 50
```

In [ ]:
```
df_train = pd.read_csv('train_v9rqX0R.csv')
df_train.head(10)
```

Out[ ]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Yea
0	FDA15	9.300	Low Fat	0.016047	Dairy	249.8092	OUT049	199
1	DRC01	5.920	Regular	0.019278	Soft Drinks	48.2692	OUT018	200
2	FDN15	17.500	Low Fat	0.016760	Meat	141.6180	OUT049	199
3	FDX07	19.200	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	199
4	NCD19	8.930	Low Fat	0.000000	Household	53.8614	OUT013	198
5	FDP36	10.395	Regular	0.000000	Baking Goods	51.4008	OUT018	200
6	FDO10	13.650	Regular	0.012741	Snack Foods	57.6588	OUT013	198
7	FDP10	NaN	Low Fat	0.127470	Snack Foods	107.7622	OUT027	198
8	FDH17	16.200	Regular	0.016687	Frozen Foods	96.9726	OUT045	200
9	FDU28	19.200	Regular	0.094450	Frozen Foods	187.8214	OUT017	200

In [ ]:
```
df_test = pd.read_csv('test_AbJTz2l.csv')
df_test.head(10)
```

Out[ ]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Yea
0	FDW58	20.750	Low Fat	0.007565	Snack Foods	107.8622	OUT049	199
1	FDW14	8.300	reg	0.038428	Dairy	87.3198	OUT017	200
2	NCN55	14.600	Low Fat	0.099575	Others	241.7538	OUT010	199
3	FDQ58	7.315	Low Fat	0.015388	Snack Foods	155.0340	OUT017	200
4	FDY38	NaN	Regular	0.118599	Dairy	234.2300	OUT027	198
5	FDH56	9.800	Regular	0.063817	Fruits and Vegetables	117.1492	OUT046	199
6	FDL48	19.350	Regular	0.082602	Baking Goods	50.1034	OUT018	200
7	FDC48	NaN	Low Fat	0.015782	Baking Goods	81.0592	OUT027	198
8	FDN33	6.305	Regular	0.123365	Snack Foods	95.7436	OUT045	200
9	FDA36	5.985	Low Fat	0.005698	Baking Goods	186.8924	OUT017	200

In [ ]:
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Item_Identifier            8523 non-null   object
 1   Item_Weight                7060 non-null   float64
 2   Item_Fat_Content           8523 non-null   object
 3   Item_Visibility            8523 non-null   float64
 4   Item_Type                  8523 non-null   object
 5   Item_MRP                   8523 non-null   float64
 6   Outlet_Identifier          8523 non-null   object
 7   Outlet_Establishment_Year  8523 non-null   int64
 8   Outlet_Size                6113 non-null   object
 9   Outlet_Location_Type       8523 non-null   object
 10  Outlet_Type                8523 non-null   object
 11  Item_Outlet_Sales          8523 non-null   float64
dtypes: float64(4), int64(1), object(7)
memory usage: 799.2+ KB
```

In [ ]:
```
df_train.nunique()
```

Out[ ]:
```
Item_Identifier              1559
Item_Weight                   415
Item_Fat_Content                5
Item_Visibility              7880
Item_Type                      16
Item_MRP                     5938
Outlet_Identifier              10
Outlet_Establishment_Year       9
Outlet_Size                     3
Outlet_Location_Type            3
Outlet_Type                     4
Item_Outlet_Sales            3493
dtype: int64
```

In [ ]:
```
df_train.isna().sum()
```

Out[ ]:
```
Item_Identifier                 0
Item_Weight                  1463
Item_Fat_Content                0
Item_Visibility                 0
Item_Type                       0
Item_MRP                        0
Outlet_Identifier               0
Outlet_Establishment_Year       0
Outlet_Size                  2410
Outlet_Location_Type            0
Outlet_Type                     0
Item_Outlet_Sales               0
dtype: int64
```

In [ ]:
```
df_train.corr(numeric_only=True)
```

Out[ ]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales
Item_Weight	1.000000	-0.014048	0.027141	-0.011588	0.014123
Item_Visibility	-0.014048	1.000000	-0.001315	-0.074834	-0.128625
Item_MRP	0.027141	-0.001315	1.000000	0.005020	0.567574
Outlet_Establishment_Year	-0.011588	-0.074834	0.005020	1.000000	-0.049135
Item_Outlet_Sales	0.014123	-0.128625	0.567574	-0.049135	1.000000

Preprocess Outlets Data

Minmax Scale Outlet_Establishment_Year

onehot encode Outlet_Identifier

create new column: is_supermarket

onehot encode supermarket types 1, 2, 3

label encode Outlet_Location_Type : Tier 1,2,3

label encode outlet size (note: has missing values)

Use Machine Learning to predict missing values for Outlet_Size

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
```

In [ ]:
```
minmax = MinMaxScaler(feature_range=(0,1))
le = LabelEncoder()
```

creating new dataframe containing only features related to the Outlet

In [ ]:
```
outlet_df = df_train.iloc[:,6:-1].copy()
outlet_df
```

Out[ ]:

	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type
0	OUT049	1999	Medium	Tier 1	Supermarket Type1
1	OUT018	2009	Medium	Tier 3	Supermarket Type2
2	OUT049	1999	Medium	Tier 1	Supermarket Type1
3	OUT010	1998	NaN	Tier 3	Grocery Store
4	OUT013	1987	High	Tier 3	Supermarket Type1
...	...	...	...	...	...
8518	OUT013	1987	High	Tier 3	Supermarket Type1
8519	OUT045	2002	NaN	Tier 2	Supermarket Type1
8520	OUT035	2004	Small	Tier 2	Supermarket Type1
8521	OUT018	2009	Medium	Tier 3	Supermarket Type2
8522	OUT046	1997	Small	Tier 1	Supermarket Type1

8523 rows × 5 columns

Defining necessary functions in one place

In [ ]:
```
def swap_sizes(x):
    if x==1:
        return 3
    elif x == 3:
        return 1
    elif x == 2:
        return 2
    else:
        return None
```

In [ ]:
```
def encode_outlet_types(dataframe):
    dataframe['is_supermarket'] = ((dataframe['Outlet_Type'] != 'Grocery Store')).astype(int)
    dataframe['SM_type1'] = ((dataframe['Outlet_Type'] == 'Supermarket Type1')).astype(int)
    dataframe['SM_type2'] = ((dataframe['Outlet_Type'] == 'Supermarket Type2')).astype(int)
    dataframe['SM_type3'] = ((dataframe['Outlet_Type'] == 'Supermarket Type3')).astype(int)
    dataframe.drop(columns=['Outlet_Type'], inplace=True)

def minmax_scale_year(dataframe):
    dataframe['Outlet_Establishment_Year'] = minmax.fit_transform(dataframe[['Outlet_Establishment_Year']])

def onehot_encode_outlet_identifier(dataframe):
    dataframe['OI'] = dataframe['Outlet_Identifier']
    dataframe = pd.get_dummies(dataframe, columns = ['OI'])
    dataframe.drop(columns=['Outlet_Identifier'], inplace=True)
    return dataframe

def label_encode_outlet_size(dataframe):
    dataframe['Outlet_Size'] = le.fit_transform(dataframe['Outlet_Size']) + 1
    #dataframe['Outlet_Size'] = dataframe['Outlet_Size'].apply(swap_sizes)
    return dataframe

def label_encode_outlet_location_type(dataframe):
    dataframe['Outlet_Location_Type'] = le.fit_transform(dataframe['Outlet_Location_Type']) + 1
```

In [ ]:
```
def fill_missing_outlet_types(dataframe, predicted_values):
    index = 0
    for i in range(0, len(dataframe)):
        item = dataframe['Outlet_Size'][i]
        if pd.isna(item):
            dataframe['Outlet_Size'][i] = predicted_values[index]
            index += 1
```

In [ ]:
```
# creates new is_supermarket column and onehot encodes the supermarket types
encode_outlet_types(outlet_df)

# minmax scales the established year column from 0 to 1
minmax_scale_year(outlet_df)

# onehot encodes the unique outlet identifier columns (10 new columns)
outlet_df = onehot_encode_outlet_identifier(outlet_df)

# encoding Outlet location types Tier 1, Tier2, Tier3 as 1,2 & 3 respectively
label_encode_outlet_location_type(outlet_df)

# encoding Outlet sizes small, medium, High as 1,2 & 3 respectively
outlet_df = label_encode_outlet_size(outlet_df)
outlet_df['Outlet_Size'] = outlet_df['Outlet_Size'].apply(swap_sizes)
```

In [ ]:
```
outlet_df
```

Out[ ]:

	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	is_supermarket	SM_type1	SM_type2	SM_type3	OI_OUT010
0	0.583333	2.0	1	1	1	0	0	0
1	1.000000	2.0	3	1	0	1	0	0
2	0.583333	2.0	1	1	1	0	0	0
3	0.541667	NaN	3	0	0	0	0	1
4	0.083333	3.0	3	1	1	0	0	0
...	...	...	...	...	...	...	...	...
8518	0.083333	3.0	3	1	1	0	0	0
8519	0.708333	NaN	2	1	1	0	0	0
8520	0.791667	1.0	2	1	1	0	0	0
8521	1.000000	2.0	3	1	0	1	0	0
8522	0.500000	1.0	1	1	1	0	0	0

8523 rows × 17 columns

Creating model to predict Outlet Sizes

creating testing and training datasets

In [ ]:
```
outlet_train = outlet_df[outlet_df['Outlet_Size'].notna()].copy()
outlet_size_to_predict = outlet_df[outlet_df['Outlet_Size'].isna()].copy()
outlet_size_input_to_predict = outlet_size_to_predict.drop(columns=['Outlet_Size'])
```

In [ ]:
```
x_outlet = outlet_train.drop(columns=['Outlet_Size'])
y_outlet = outlet_train['Outlet_Size']
```

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
```

In [ ]:
```
x_train_outlet, x_val_outlet, x_test_outlet, y_val_outlet = train_test_split(x_outlet, y_outlet, test_size=0.35
```

In [ ]:
```
lr_out = LogisticRegression()
lr_out.fit(x_train_outlet, x_test_outlet)
y_val_pred_out = lr_out.predict(x_val_outlet)
accuracy_score(y_val_outlet, y_val_pred_out)
```

Out[ ]:
```
1.0
```

Model gives a perfect accuracy score of 1

In [ ]:
```
predicted_outlet_sizes = lr_out.predict(outlet_size_input_to_predict)
predicted_outlet_sizes
```

Out[ ]:
```
array([2., 1., 1., ..., 2., 1., 1.])
```

Fill the missing outlet sizes in outlet_df sequentially and convert from float to integer column

In [ ]:
```
fill_missing_outlet_types(outlet_df, predicted_outlet_sizes)
outlet_df['Outlet_Size'] = outlet_df['Outlet_Size'].astype('int')
```

Fully processed Outlet data

In [ ]:
```
outlet_df
```

Out[ ]:

	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	is_supermarket	SM_type1	SM_type2	SM_type3	OI_OUT010
0	0.583333	2	1	1	1	0	0	0
1	1.000000	2	3	1	0	1	0	0
2	0.583333	2	1	1	1	0	0	0
3	0.541667	2	3	0	0	0	0	1
4	0.083333	3	3	1	1	0	0	0
...	...	...	...	...	...	...	...	...
8518	0.083333	3	3	1	1	0	0	0
8519	0.708333	1	2	1	1	0	0	0
8520	0.791667	1	2	1	1	0	0	0
8521	1.000000	2	3	1	0	1	0	0
8522	0.500000	1	1	1	1	0	0	0

8523 rows × 17 columns

Creating outlet dataset for testing set

In [ ]:
```
outlet_df_test = df_test.iloc[:,6:].copy()
outlet_df_test
```

Out[ ]:

	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type
0	OUT049	1999	Medium	Tier 1	Supermarket Type1
1	OUT017	2007	NaN	Tier 2	Supermarket Type1
2	OUT010	1998	NaN	Tier 3	Grocery Store
3	OUT017	2007	NaN	Tier 2	Supermarket Type1
4	OUT027	1985	Medium	Tier 3	Supermarket Type3
...	...	...	...	...	...
5676	OUT046	1997	Small	Tier 1	Supermarket Type1
5677	OUT018	2009	Medium	Tier 3	Supermarket Type2
5678	OUT045	2002	NaN	Tier 2	Supermarket Type1
5679	OUT017	2007	NaN	Tier 2	Supermarket Type1
5680	OUT045	2002	NaN	Tier 2	Supermarket Type1

5681 rows × 5 columns

Applying the same functions previously applied on the training set

In [ ]:
```
encode_outlet_types(outlet_df_test)
minmax_scale_year(outlet_df_test)
outlet_df_test = onehot_encode_outlet_identifier(outlet_df_test)
label_encode_outlet_location_type(outlet_df_test)
outlet_df_test = label_encode_outlet_size(outlet_df_test)
outlet_df_test['Outlet_Size'] = outlet_df_test['Outlet_Size'].apply(swap_sizes)
```

Filling missing outlet_sizes. From the observations in the training dataset, the outlet_sizes for the outlet identifiers are: OUT045 = 1, OUT017 = 1, OUT010 = 2

In [ ]:
```
outlet_df_test.loc[outlet_df_test['OI_OUT010']==1, 'Outlet_Size'] = 2
outlet_df_test.loc[outlet_df_test['OI_OUT017']==1, 'Outlet_Size'] = 1
outlet_df_test.loc[outlet_df_test['OI_OUT045']==1, 'Outlet_Size'] = 1
outlet_df_test['Outlet_Size'] = outlet_df_test['Outlet_Size'].astype('int')
```

Fully processed outlet testing dataset

In [ ]:
```
outlet_df_test
```

Out[ ]:

	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	is_supermarket	SM_type1	SM_type2	SM_type3	OI_OUT010
0	0.583333	2	1	1	1	0	0	0
1	0.916667	1	2	1	1	0	0	0
2	0.541667	2	3	0	0	0	0	1
3	0.916667	1	2	1	1	0	0	0
4	0.000000	2	3	1	0	0	1	0
...	...	...	...	...	...	...	...	...
5676	0.500000	1	1	1	1	0	0	0
5677	1.000000	2	3	1	0	1	0	0
5678	0.708333	1	2	1	1	0	0	0
5679	0.916667	1	2	1	1	0	0	0
5680	0.708333	1	2	1	1	0	0	0

5681 rows × 17 columns

Preprocess Items Data

Label encode Item_Fat_Content low fat and regular fat as 0 and 1 respectively

onehot encode Item_Type

Fill the missing values in Item_Weight. The same Item_Identifier should have the same Item_Weight

Remove outliers in Item_weight, Item_Visibility and Item_MRP (only when training)

In [ ]:
```
items_df = df_train.iloc[:,:6].copy()
items_df
```

Out[ ]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP
0	FDA15	9.300	Low Fat	0.016047	Dairy	249.8092
1	DRC01	5.920	Regular	0.019278	Soft Drinks	48.2692
2	FDN15	17.500	Low Fat	0.016760	Meat	141.6180
3	FDX07	19.200	Regular	0.000000	Fruits and Vegetables	182.0950
4	NCD19	8.930	Low Fat	0.000000	Household	53.8614
...	...	...	...	...	...	...
8518	FDF22	6.865	Low Fat	0.056783	Snack Foods	214.5218
8519	FDS36	8.380	Regular	0.046982	Baking Goods	108.1570
8520	NCJ29	10.600	Low Fat	0.035186	Health and Hygiene	85.1224
8521	FDN46	7.210	Regular	0.145221	Snack Foods	103.1332
8522	DRG01	14.800	Low Fat	0.044878	Soft Drinks	75.4670

8523 rows × 6 columns

Save corresponding item weights for each item identifier as a dictionary

In [ ]:
```
item_weight_mappings = {}

for i in range(0, len(items_df)):
    item = items_df['Item_Identifier'][i]
    weight = items_df['Item_Weight'][i]
    if not pd.isna(weight):
        item_weight_mappings[item] = weight
```

Define necessary functions in one place

In [ ]:
```
def encode_item_fat_content(dataframe):
    dataframe['Item_Fat_Content'] = ((dataframe['Item_Fat_Content']=='Regular') | (dataframe['Item_Fat_Content

def fill_missing_item_types(dataframe):
    for i in range(0, len(dataframe)):
```

```
        item = dataframe['Item_Identifier'][i]
        weight = dataframe['Item_Weight'][i]
        if pd.isna(weight):
            if item in item_weight_mappings.keys():
                dataframe['Item_Weight'][i] = item_weight_mappings[item]

def onehot_encode_item_types(dataframe):
    dataframe = pd.get_dummies(dataframe, columns = ['Item_Type'])
    return dataframe
```

In [ ]:
```
len(list(item_weight_mappings.keys()))
```

Out[ ]:
```
1555
```

There are four mappings missing. So the identifier does not have a default weight to assign to

In [ ]:
```
# Label encode Item_Fat_Content low fat and regular fat as 0 and 1 respectively
encode_item_fat_content(items_df)

# Fill the missing values in Item_Weight based on the mapping created earlier
fill_missing_item_types(items_df)

# onehot encoding Item_Types
items_df = onehot_encode_item_types(items_df)
```

To deal with the four identifiers with no default weight

In [ ]:
```
unique_item_weights = (items_df.groupby(['Item_Identifier'])['Item_Weight'].nunique() == 0).to_dict()
have_no_weight_to_assign = []
for key in unique_item_weights:
    if unique_item_weights[key] == True:
        have_no_weight_to_assign.append(key)
have_no_weight_to_assign
```

Out[ ]:
```
['FDE52', 'FDK57', 'FDN52', 'FDQ60']
```

In [ ]:
```
items_with_no_weight = items_df[(items_df['Item_Identifier'] == 'FDE52') | (items_df['Item_Identifier'] == 'FDK
items_with_no_weight
```

Out[ ]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Item_Type_Baking Goods	Item_Type_Breads	Item_Type_E
927	FDN52	NaN	1	0.130933	86.9198	0	0	
1922	FDK57	NaN	0	0.079904	120.0440	0	0	
4187	FDE52	NaN	1	0.029742	88.9514	0	0	
5022	FDQ60	NaN	1	0.191501	121.2098	1	0	

Filling the four missing rows with the average weight of their corresponding Item type. Also adding them to the weight mapping

In [ ]:
```
frozed_food_avg = round(items_df.loc[items_df['Item_Type_Frozen Foods'] == 1, "Item_Weight"].mean(), 3)
snack_foods_avg = round(items_df.loc[items_df['Item_Type_Snack Foods'] == 1, "Item_Weight"].mean(), 3)
dairy_avg = round(items_df.loc[items_df['Item_Type_Dairy'] == 1, "Item_Weight"].mean(), 3)
baking_goods_avg = round(items_df.loc[items_df['Item_Type_Baking Goods'] == 1, "Item_Weight"].mean(), 3)

items_df['Item_Weight'][927] = frozed_food_avg
items_df['Item_Weight'][1922] = snack_foods_avg
items_df['Item_Weight'][4187] = dairy_avg
items_df['Item_Weight'][5022] = baking_goods_avg

item_weight_mappings['FDN52'] = frozed_food_avg
item_weight_mappings['FDK57'] = snack_foods_avg
item_weight_mappings['FDE52'] = dairy_avg
item_weight_mappings['FDQ60'] = baking_goods_avg
```

Fully processed items training dataset

In [ ]:
```
items_df
```

Out[ ]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Item_Type_Baking Goods	Item_Type_Breads	Item_Type_E
0	FDA15	9.300	0	0.016047	249.8092	0	0	
1	DRC01	5.920	1	0.019278	48.2692	0	0	
2	FDN15	17.500	0	0.016760	141.6180	0	0	
3	FDX07	19.200	1	0.000000	182.0950	0	0	
4	NCD19	8.930	0	0.000000	53.8614	0	0	
...	...	...	...	...	...	...	...	
8518	FDF22	6.865	0	0.056783	214.5218	0	0	
8519	FDS36	8.380	1	0.046982	108.1570	1	0	
8520	NCJ29	10.600	0	0.035186	85.1224	0	0	
8521	FDN46	7.210	1	0.145221	103.1332	0	0	
8522	DRG01	14.800	0	0.044878	75.4670	0	0	

8523 rows × 21 columns

Preparing testing dataset

In [ ]:
```
items_df_test = df_test.iloc[:,:6].copy()
```

Applying the same functions we did in the training dataset

In [ ]:
```
encode_item_fat_content(items_df_test)
fill_missing_item_types(items_df_test)
items_df_test = onehot_encode_item_types(items_df_test)
```

In [ ]:
```
items_df_test
```

Out[ ]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Item_Type_Baking Goods	Item_Type_Breads	Item_Type_E
0	FDW58	20.750	0	0.007565	107.8622	0	0	
1	FDW14	8.300	1	0.038428	87.3198	0	0	
2	NCN55	14.600	0	0.099575	241.7538	0	0	
3	FDQ58	7.315	0	0.015388	155.0340	0	0	
4	FDY38	13.600	1	0.118599	234.2300	0	0	
...	...	...	...	...	...	...	...	
5676	FDB58	10.500	1	0.013496	141.3154	0	0	
5677	FDD47	7.600	1	0.142991	169.1448	0	0	
5678	NCO17	10.000	0	0.073529	118.7440	0	0	
5679	FDJ26	15.300	1	0.000000	214.6218	0	0	
5680	FDU37	9.500	1	0.104720	79.7960	0	0	

5681 rows × 21 columns

In [ ]:
```
print(items_df.isna().sum().any())
print(items_df_test.isna().sum().any())
```

```
False
False
```

Merging Items and Outlet datasets

In [ ]:
```
final_train_df = pd.concat((items_df, outlet_df), axis=1)
final_train_df
```

Out[ ]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Item_Type_Baking Goods	Item_Type_Breads	Item_Type_E
0	FDA15	9.300	0	0.016047	249.8092	0	0	
1	DRC01	5.920	1	0.019278	48.2692	0	0	
2	FDN15	17.500	0	0.016760	141.6180	0	0	
3	FDX07	19.200	1	0.000000	182.0950	0	0	
4	NCD19	8.930	0	0.000000	53.8614	0	0	
...	...	...	...	...	...	...	...	
8518	FDF22	6.865	0	0.056783	214.5218	0	0	
8519	FDS36	8.380	1	0.046982	108.1570	1	0	
8520	NCJ29	10.600	0	0.035186	85.1224	0	0	
8521	FDN46	7.210	1	0.145221	103.1332	0	0	
8522	DRG01	14.800	0	0.044878	75.4670	0	0	

8523 rows × 38 columns

In [ ]:
```
final_test_df = pd.concat((items_df_test, outlet_df_test), axis=1)
final_test_df
```

Out[ ]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Item_Type_Baking Goods	Item_Type_Breads	Item_Type_E
0	FDW58	20.750	0	0.007565	107.8622	0	0	
1	FDW14	8.300	1	0.038428	87.3198	0	0	
2	NCN55	14.600	0	0.099575	241.7538	0	0	
3	FDQ58	7.315	0	0.015388	155.0340	0	0	
4	FDY38	13.600	1	0.118599	234.2300	0	0	
...	...	...	...	...	...	...	...	
5676	FDB58	10.500	1	0.013496	141.3154	0	0	
5677	FDD47	7.600	1	0.142991	169.1448	0	0	
5678	NCO17	10.000	0	0.073529	118.7440	0	0	
5679	FDJ26	15.300	1	0.000000	214.6218	0	0	
5680	FDU37	9.500	1	0.104720	79.7960	0	0	

5681 rows × 38 columns

Feature Selection based on correlation

In [ ]:
```
X_train_df = final_train_df.iloc[:, [1,2,3,4,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37]].copy()
X_test_df = final_test_df.iloc[:, [1,2,3,4,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37]].copy()
Y_train_all = df_train['Item_Outlet_Sales']
```

In [ ]:
```
X_train_df.head()
```

Out[ ]:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Size	Outlet_Location_Type	is_supermarket	SM_type1	SM_ty
0	9.30	0	0.016047	249.8092	2	1	1	1	
1	5.92	1	0.019278	48.2692	2	3	1	0	
2	17.50	0	0.016760	141.6180	2	1	1	1	
3	19.20	1	0.000000	182.0950	2	3	0	0	
4	8.93	0	0.000000	53.8614	3	3	1	1	

Minmax scaling some numeric columns

In [ ]:
```
X_train_df[['Item_Weight','Item_MRP','Item_Visibility']] = minmax.fit_transform(X_train_df[['Item_Weight','Iter
X_test_df[['Item_Weight','Item_MRP','Item_Visibility']] = minmax.transform(X_test_df[['Item_Weight','Item_MRP'
```

In [ ]:
```
X_train, X_val, y_train, y_val = train_test_split(X_train_df, Y_train_all, test_size=0.2, random_state=13)
```

Model Training and testing on testing set itself

In [ ]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

Using GradientBoostingRegressor Model

All Hyperparams : min_samples_split, min_samples_leaf, min_weight_fraction_leaf, max_depth, max_leaf_nodes, max_features, learning_rate, n_estimators, subsample, loss, init

We'll select just 3

In [ ]:
```
param_grid = {
    'learning_rate': [0.15, 0.16, 0.17, 0.18],
    'max_depth' : [2,3,4],
    'n_estimators':[35, 40, 45, 50, 55]
}
gbr_gscv = GradientBoostingRegressor()
gbr_gscv = GridSearchCV(gbr_gscv, param_grid=param_grid, n_jobs=1, scoring='neg_mean_absolute_error', cv=5)
gbr_gscv.fit(X_train, y_train)
```

Out[ ]:
```
GridSearchCV
estimator: GradientBoostingRegressor
GradientBoostingRegressor
```

In [ ]:
```
print(gbr_gscv.best_params_)
print(gbr_gscv.best_score_)
```

```
{'learning_rate': 0.16, 'max_depth': 3, 'n_estimators': 35}
-759.3721914384265
```

The best hyperparameters are learning_rate=0.16, max_depth=3, n_estimators=35

In [ ]:
```
Gbr = GradientBoostingRegressor(learning_rate=0.16, max_depth=3, n_estimators=35)
Gbr.fit(X_train, y_train)
y_pred_gbr = Gbr.predict(X_val)
```

In [ ]:
```
print("Mean Absolute Error :", mean_absolute_error(y_val, y_pred_gbr))
print("Root Mean Squared Error :", mean_squared_error(y_val, y_pred_gbr) ** 0.5)
```

```
Mean Absolute Error : 753.9049619538758
Root Mean Squared Error : 1080.5526756498043
```

Training using Entire training dataset

In [ ]:
```
Final_model = GradientBoostingRegressor(learning_rate=0.16, max_depth=3, n_estimators=35)
Final_model.fit(X_train_df, Y_train_all)
```

Out[ ]:
```
GradientBoostingRegressor
GradientBoostingRegressor(learning_rate=0.16, n_estimators=35)
```

Making final Predictions

In [ ]:
```
final_predictions = Final_model.predict(X_test_df)
```

In [ ]:
```
df_sol = pd.read_csv('sample_submission_8RXa3c6.csv')
df_sol.head()
```

Out[ ]:

	Item_Identifier	Outlet_Identifier	Item_Outlet_Sales
0	FDW58	OUT049	1000
1	FDW14	OUT017	1000
2	NCN55	OUT010	1000
3	FDQ58	OUT017	1000
4	FDY38	OUT027	1000

In [ ]:
```
df_submission = df_sol.copy()
```

In [ ]:
```
df_submission['Item_Outlet_Sales'] = final_predictions
df_submission
```

Out[ ]:

	Item_Identifier	Outlet_Identifier	Item_Outlet_Sales
0	FDW58	OUT049	1658.538588
1	FDW14	OUT017	1384.366405
2	NCN55	OUT010	638.209518
3	FDQ58	OUT017	2448.561692
4	FDY38	OUT027	6189.772366
...	...	...	...
5676	FDB58	OUT046	2176.413464
5677	FDD47	OUT018	2471.889494
5678	NCO17	OUT045	1816.102572
5679	FDJ26	OUT017	3578.181005
5680	FDU37	OUT045	1281.159493

5681 rows × 3 columns

Checking if there's any negative or zero values

In [ ]:
```
(df_submission['Item_Outlet_Sales']<=0).any()
```

Out[ ]:
```
False
```

Exporting file to CSV

In [ ]:
```
df_submission.to_csv('final_submission.csv', index=False)
```

Analytics Vidya Submission and Score